Detect Large Language Model (LLM) generated text using Naive Bayes

Prabhleensaluja
5 min read · Just now

With increase in the level of sophistication of the text generated using AI these days it is getting difficult to differentiate AI generated text from human written essays. The objective of this competition is to use Natural Language Tool Kit (NLTK) and Machine Learning methods to identify AI generated essays given the 2 prompts in the dataset.

0	Car-free cities
1	Does the electoral college work?

Data:-

The data contains 1378 records of essays given the 2 prompts ‘Car-free cities’ and ‘Does the electoral college work’. The main issue with this dataset was it only contained 3 essays generated using Large Language Model (LLM) and the rest are all human written essays. For this dataset is modified and now it contains 1820 records in total, that contains 445 LLM generated essays and 1375 human written essays.

```
len(df)
```

```
1820
```

```
len(df[df.generated==0])
```

```
1375
```

```
len(df[df.generated==1])
```

```
445
```

This dataset is further going to be split into 2datasets:-

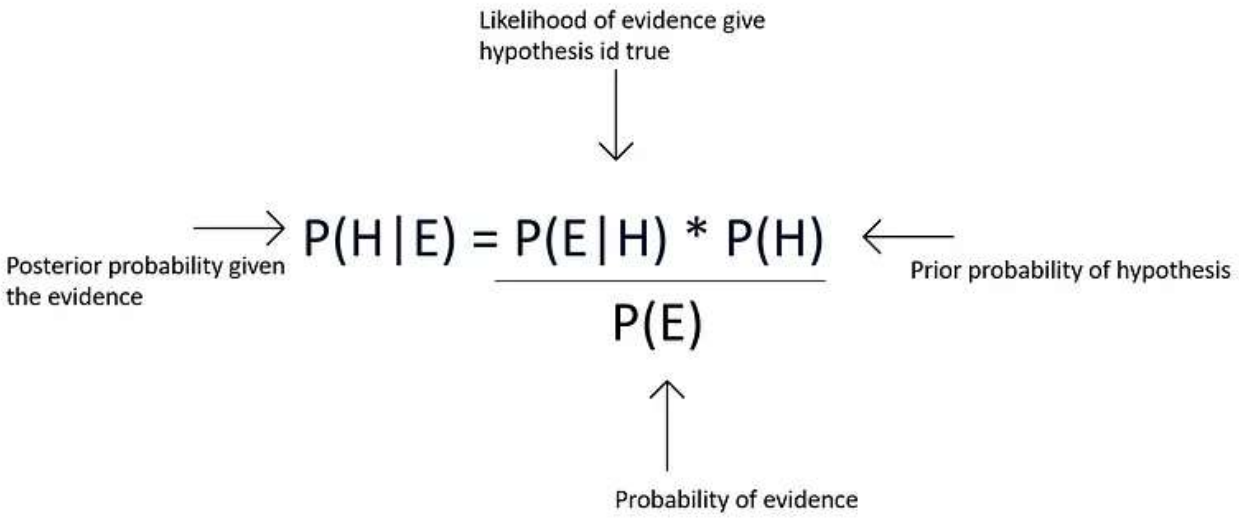
```
df_train, df_dev = train_test_split(df, test_size = 0.2, random_state = 10)
```

1. **Train:-** This dataset will be used to create and train our model and is the largest portion amongst the 2.
2. **Dev:-** This dataset is useful for choosing a value for hyperparameters that gives us the best performance. (In our case this hyperparameter is the smoothing parameter 'alpha')

My Kaggle and GitHub

Naive Bayes Classifier

Naive Bayes Classifier is a supervised Machine Learning algorithm that can be used to predict the likelihood of a datapoint belonging to a particular class. Naive Bayes gets it's name from the Naive assumption that it makes that all the features of a datapoint are completely independent of each other which makes probability calculations much easier. The Naive Bayes classifier relies on the principles of Bayes' theorem which is one of the most important theorem that gives us the probability of an event (hypothesis) occurring given another event (evidence) has already occurred. The theorem can be stated as follows:-



Data preprocessing:-

As the dataset comprises textual information, we tokenize the data, eliminate stop words, and apply stemming to the tokens.

```
def preprocess_data(data_row):
    print(data_row)
    stop_words = set(stopwords.words('english'))
    tokens = [word.lower() for word in nltk.word_tokenize(data_row['text']) if word.isalpha() and word.lower() not in stop_words]
    # Stemming
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in tokens]

    return stemmed_tokens
```

The model also uses count vectorizer to create a vocabulary with the threshold as minimum 5 occurrences to be a part of the vocabulary.

```
from nltk import word_tokenize
# Threshold is set to 5
vocab = {}
vectorizer = CountVectorizer(tokenizer=word_tokenize, min_df=6)

# Fit and transform the data
X = vectorizer.fit_transform(df_train['text'])

# Get the feature names (words in the vocabulary)
feature_names = vectorizer.get_feature_names_out()

# Print the vocabulary and corresponding counts
for feature, count in zip(feature_names, X.sum(axis=0).A1):
    vocab[feature] = count

print(vocab)
word2idx = {w: i for i, w in enumerate(vocab)}
```

Model creation:-

Model creation consists of the following steps:-

1. **Calculating the prior probability:-** Calculating the probability human written essay or LLM generated essay is very simple, it just includes dividing the number of human written essays or number of LLM generated essays by total number of essays that contain the particular word.

```
essays = len(df_train['text'])
human_essays = len(df_train[df_train['generated'] == 0])
llm_essays = len(df_train[df_train['generated'] == 1])
p_human = human_essays / essays
p_llm = llm_essays / essays
print(p_human)
print(p_llm)
```

2. Counting the number of documents that contain the respective word:- First we are going to calculate the number of human written and number of LLM generated that contain each words from the vocab.

```
word_counts_human = np.zeros(len(vocab), dtype=int)
word_counts_llm = np.zeros(len(vocab), dtype=int)
for index, row in df_train.iterrows():
    tokens = row["tokens"]
    if row["generated"] == 0:
        word_counts_human[[word2idx[w] for w in tokens if w in vocab]] += 1
    else:
        word_counts_llm[[word2idx[w] for w in tokens if w in vocab]] += 1
```

3. **Calculating posterior probabilities:-** Then further we calculate the log likelihood of human written and LLM generated essays containing a particular word

```
def calculate_probabilities(alpha):
    p_word_human = (word_counts_human + alpha) / (human_essays + alpha * len(vocab))
    p_word_llm = (word_counts_llm + alpha) / (llm_essays + alpha * len(vocab))
    return p_word_human, p_word_llm
```

4. **Using above probabilities to predict the class of an essay:-** Here we are using the posterior probabilities and smoothing parameters to calculate the log likelihood of an essay being human generated and LLM generated essay. If the log likelihood of the essay being human generated we return 0 (human generated) as class, otherwise we return 1 (LLM generated)

```
def predict(essay, alpha):
    p_word_human, p_word_llm = calculate_probabilities(alpha)
    words = essay.lower().split()

    # Calculate the log-likelihood of each class
    log_p_human = np.log(p_human) + np.sum(np.log(p_word_human[[word2idx[w] for w in words if w in vocab]]))
    log_p_llm = np.log(p_llm) + np.sum(np.log(p_word_llm[[word2idx[w] for w in words if w in vocab]]))

    # Return the predicted class
    if log_p_human > log_p_llm:
        return 0
    else:
        return 1
```

Model Evaluation:-

```
def evaluate(df, alpha):
    y_true = df["generated"]
    y_pred = df["text"].apply(predict, alpha = alpha)

    tp = ((y_true == 0) & (y_pred == 0)).sum()
    fp = ((y_true == 1) & (y_pred == 0)).sum()
    fn = ((y_true == 0) & (y_pred == 1)).sum()
    tn = ((y_true == 1) & (y_pred == 1)).sum()

    accuracy = (tp + tn) / len(df)
    precision = tp / (tp + fp)
    recall = tp / (tp + fn)
    f1_score = 2 * precision * recall / (precision + recall)

    # Calculate accuracy
    acc_score = accuracy_score(y_true, y_pred)

    return accuracy, precision, recall, f1_score, acc_score

# Evaluate the NBC on the development set
alpha_values = [0.1, 0.3, 0.5, 0.8, 1, 1.3, 1.5]
best_accuracy = 0
best_alpha = 0
for alpha in alpha_values:
    accuracy, precision, recall, f1_score, acc_score = evaluate(df_test, alpha)
    if(acc_score > best_accuracy):
        best_accuracy = accuracy
        best_alpha = alpha
```


dataset which gives us the best real world accuracy.

```
Best Accuracy_score: 0.90
Alpha: 0.1
```

In this case I get best accuracy (90%) when alpha (smoothing parameter) is 0.1

Contribution:-

1. The dataset contained very few Large Language Model (LLM) generated essays which would not enough to give us any useful information for prediction. I added 423 LLM generated essays in total for the 2 given prompts 'Car- free cities' and 'Does the electoral college work' which gave us enough data for prediction.
2. The only hyperparameter used here is the smoothing parameter (alpha) for which I have used the dev dataset to select the best smoothing parameter here I am looping through the following array of smoothing parameters and using the dev dataset to find which one gives us the best accuracy.

References:-

1. https://asa5604.wixsite.com/datamining/post/template-the-ultimate-guide-to-writing-the-ultimate-guide
2. https://machinelearningmastery.com/naive-bayes-classifier-scratch-python/
3. https://medium.com/@rangavamsi5/na%C3%AFve-bayes-algorithm-implementation-from-scratch-in-python-7b2cc39268b9

4. https://towardsdatascience.com/implementing-naive-bayes-algorithm-from-scratch-python-c6880cfc9c41

5. https://www.kaggle.com/competitions/llm-detect-ai-generated-text?rvi=1

Large Language Models | AI | Naive Bayes | Naive Bayes Classifier | Text Classification

Written by Prabhleensaluja

Edit profile

0 Followers

Recommended from Medium

Rohit Verma

My Interview Experience at Google [L5 Offer]

Comprehensive Insights: A Deep Dive into the Journey from Preparation Through Interview...

9 min read · Nov 25

904 18

Eva Lesko Natiello in The Writing Cooperative

How a Professor's One Piece of Advice Influenced My Entire...

As a new writer and student in mid-life, unexpected lessons about creativity...

5 min read · 5 days ago

5.6K 119

Lists

Generative AI Recommended Reading

52 stories · 465 saves

What is ChatGPT?

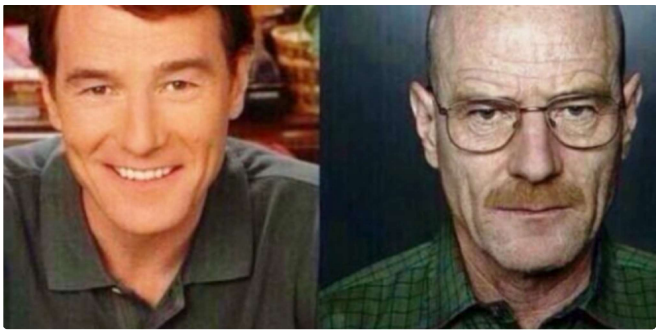
9 stories · 239 saves

The New Chatbots: ChatGPT, Bard, and Beyond

12 stories · 226 saves

Natural Language Processing

924 stories · 441 saves



David Goudet

This is Why I Didn't Accept You as a Senior Software Engineer

An Alarming Trend in The Software Industry

5 min read · Jul 25

3.6K 49

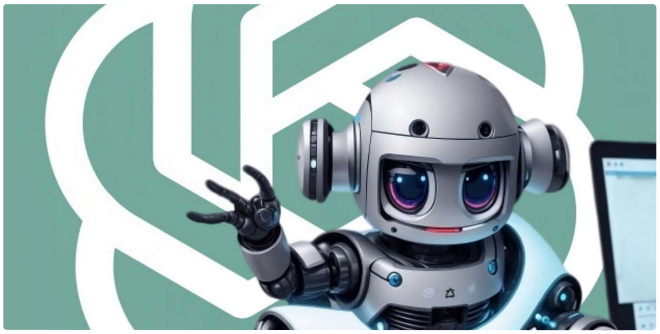
Nick Wignall

5 Signs of Low Emotional Maturity

#2: Gaslighting

9 min read · 3 days ago

1.8K 54



Neeramitra Reddy in The Startup

3 Advanced (and Unique) ChatGPT Uses You've Likely Not Seen Before

Valuable "meta" use cases I've found in 10 months of tinkering with ChatGPT

11 min read · 5 days ago

3.8K 55

Ankit Detroja

Unveiling JavaScript: The Double-Bang (!!) Operator

What is the Double-Bang Operator?

1 min read · 2 days ago

65 1

See more recommendations